**Finding the Location Furthest from Water in the Conterminous United States**

The idea for this post came a few months back when I received an email  
that started, “I am a writer and teacher and am reaching out to you with  
a question related to a piece I would like to write about the place in  
the United States that is furthest from a natural body of surface water.  
My question is, in short, might you know where this theoretical place is  
located?”

As someone who works on the  
[National Water Census](https://water.usgs.gov/watercensus/)  
and  
[National hydrography data](https://www.usgs.gov/core-science-systems/ngp/national-hydrography),  
this question piqued my interest. I also had an idea of one way to  
answer the question that would use some tools and techniques I’ve  
recently been developing in R. What follows is just that, **one** way  
that this question might be answered. The code I used to find the  
solution is presented in detail after the solution to the question.

The solution graphics shown below were generated with the following  
code: The furthest\_water.R script can be  
[found here](https://github.com/USGS-OWI/owi-blog/tree/master/content/static/furthest-water).

source("furthest\_water.R")

furthest\_water(scenario = "waterbodies")

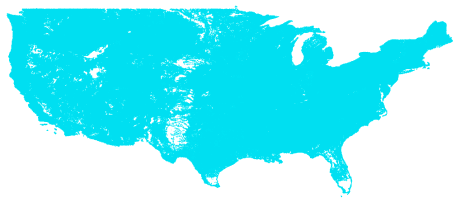
furthest\_water(scenario = c("waterbodies", "filter\_monthly\_flow"))

furthest\_water(scenario = c("waterbodies", "remove\_intermittent", "filter\_monthly\_flow"))

**Method**

To answer the question, “where is the furthest place from a natural body  
of water?”, I first had to decide where all the places I might want to  
check are. To do this, I first built a search box around the  
southwestern US to limit the data volume I had to process. I then  
created a grid of points 5km apart within the search box and the US  
coastline. This set of points, pictured below, will stand in for all the  
places we might want to look. Note that there are actually about 150k  
points on this image, but they draw as one area because their symbols  
overlap.

All Search
Points  
With all possible places defined, I needed all the water bodies. The  
National Hydrography Dataset contains data used to map rivers and lakes  
across the country. As a bonus, there is a dataset known as the “NHDPlus  
v2.1 National Seamless” that is available as a single geospatial  
database for the whole conterminous US. See  
[this EPA page](https://www.epa.gov/waterdata/get-data)  
for lots of helpful documentation and data downloads. The NHDPlus  
contains a few million river segments and almost a half million water  
bodies mapped at 1:100,000-scale. The NHDPlus also has some useful  
attributes that categorize waterbodies as ephemeral (sometimes dry) or  
perennial (always wet) and modeled estimates of mean flow by month of  
the year for rivers. Geospatial data representing rivers and bodies of  
water like lakes along with the attributes about how likely they are to  
be wet is just what I needed. The maps below show how dense the rivers  
and water bodies in the NHDPlus are.

  
All water
bodies

The algorithm I used to figure out which of the search locations is  
furthest from water uses a nearest neighbor search implemented in an  
[R package called RANN](https://cran.r-project.org/web/packages/RANN/)  
which allowed me to find which points are within a given distance of a  
flowline or water body. By starting with a small distance and  
incrementing to larger and larger distances, removing locations as I  
went, I was able to narrow down to one point that was further than any  
other from all the water bodies. The marvel of this technique is that  
it’s not just 150k points and a few million rivers and lakes. To make it  
work, I converted the lines and polygons that represent rivers and water  
bodies to more than 30 million individual points that make up all the  
geometry!

In each of the three scenarios below, there are two graphics. The first,  
is a simple visual showing where the location is relative to rivers and  
water bodies that are in the vicinity. The second is an animated graphic  
showing each step of the nearest neighbor search as the search radius  
increases incrementally.

**Furthest from where there might be water**

In the first scenario, I included everything in the NHDPlus that might  
be a water body. This includes lakes and rivers that are categorized as  
ephemeral and rivers with attributes that show they are usually dry one  
or more months a year. As you can see below, the location is in the  
Bonneville salt flats. The irony is that, while not categorized as a  
water body in the NHD, this picture taken from I-80 shows that the salt  
flats do get filled with water periodically!

By  
[Ricraider](https://commons.wikimedia.org/w/index.php?title=User:Ricraider&action=edit&redlink=1)  
– Own work,  
[CC  
BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0),  
[Link](https://commons.wikimedia.org/w/index.php?curid=30988413)

[*Explore this place in google maps.*](https://www.google.com/maps/place/32%C2%B016'52.9%22N+113%C2%B059&%2339;03.0%22W/@32.2813555,-113.9863557,705m/data=!3m2!1e3!4b1!4m6!3m5!1s0x0:0x0!7e2!8m2!3d32.2813509!4d-113.9841667)

Furthest from where there might be
water  
Furthest from where there might be water
animation

**Furthest from modeled water**

The place we say is the furthest from water shouldn’t be a dry lake bed,  
should it? Let’s look at another scenario. In this one, I used the  
monthly average flow estimates from the NHDPlus. These estimates are  
calculated using the “Enhanced Runoff Method” or EROM. You can find  
documentation  
[about the method here.](https://www.epa.gov/waterdata/learn-more)  
The EROM estimates are available for each month of the year, providing a  
normal monthly-mean flow estimate for the period of the analysis (1971  
to 2000). For this scenario, I removed any flowline that had a  
monthly-mean flow estimate of zero for any month. As can be seen below,  
I came up with a location almost on the US-Mexico border in the desert  
in far south west Arizona.

This is a place I would believe is actually the furthest from a natural  
body of water but I noticed that there were some small waterbodies in  
the desert that almost certainly dry out in the summer.

[*Explore this place in google maps*](https://www.google.com/maps/place/32%C2%B016'52.9%22N+113%C2%B059&%2339;03.0%22W/@32.2813555,-113.9863557,705m/data=!3m2!1e3!4b1!4m6!3m5!1s0x0:0x0!7e2!8m2!3d32.2813509!4d-113.9841667)

Furthest from modeled
water  
Furthest from modeled water
animation

**Furthest from modeled and non-ephemeral water**

For the third scenario, I used the same mean-monthly flow method to  
remove rivers that probably dry out for a month or more a year and I  
removed flowlines and water bodies that were categorized as ephemeral.  
This removed many water body polygons in the desert that are only wet  
part of the year. As can be seen below, now we are in the Sonoran desert  
west of Phoenix, AZ. By the looks of things, this really may be the  
furthest from a natural body of surface water in the US (over 72km/45mi)  
– especially after a long period with no rain.

[*Explore this place in google maps*](https://www.google.com/maps/place/33%C2%B032'40.5%22N+113%C2%B041&%2339;36.6%22W/@33.5445945,-113.6956887,695m/data=!3m2!1e3!4b1!4m5!3m4!1s0x0:0x0!8m2!3d33.54459!4d-113.6935)

Furthest from modeled and non-ephemeral
water  
Furthest from modeled and non-ephemeral water
animation

As the variety of results given different inputs shows, this question  
doesn’t necessarily have one answer. It depends what we assume a body of  
water is and how often that body of water needs to be wet. The factors  
at play that determine if a river or water body is wet are diverse and  
highly interrelated. Ranging from the obvious, like how much  
[it’s rained recently](https://water.weather.gov/precip/),  
to less obvious, like how much  
[groundwater is flowing into or out of a river](https://water.usgs.gov/edu/rivers-contain-groundwater.html).  
Ecosystems have a role to play too. In desert environments,  
[some plants depend on groundwater](https://pubs.usgs.gov/wsp/1423/report.pdf)  
which can actually draw down groundwater around rivers, reducing flows  
or even causing surface flow to cease even though groundwater may  
continue to be available just under the surface. This is just a brief  
discussion of the complexities of the natural water cycle and the  
natural features that arise from it. For more, the  
[USGS Water Science School](https://water.usgs.gov/edu/watercycle.html)  
has a wealth of information and all  
[USGS publications](https://pubs.er.usgs.gov/)  
and  
[data](https://data.usgs.gov/datacatalog/#fq=dataType%3A(collection%20OR%20non-collection)&fq=keywords%3A%22NWIS%22&q=*%3A*)  
are free and available for everyone.

**Code Explanation**

The three scenarios above were run using a function that executes the  
code described below. There is a mix of comments and summary text  
written outside the code. To execute this, you will need to install the  
required packages and download the  
[NHDPlus National Seamless Database](https://www.epa.gov/waterdata/nhdplus-national-data)  
and a  
[state boundaries layer available here.](https://www.arcgis.com/home/item.html?id=b07a9393ecbd430795a6f6218443dccc)  
These two datasets are referenced in the code.

**Load Packages and Data**

In this first step, I load all the packages and data I need.

Packages include:

* [sf](https://r-spatial.github.io/sf/)
* [dplyr](https://dplyr.tidyverse.org/)
* [RANN](https://cran.r-project.org/web/packages/RANN/index.html)
* [nhdplusTools](https://dblodgett-usgs.github.io/nhdplusTools/)
* [units](https://r-quantities.github.io/units/)
* [stringr](https://stringr.tidyverse.org/)
* [gifski](https://cran.r-project.org/web/packages/gifski/index.html)
* [ggmap](https://cran.r-project.org/web/packages/ggmap/)

Note that sf and dplyr do the vast majority of the work and they are  
loaded with a library() command. All other packages are accessed using  
the package::function() syntax

Variables created here are:

* flowlines: NHDPlus flowlines with lots of attributes. Loaded with  
  [nhdplusTools](https://dblodgett-usgs.github.io/nhdplusTools/)  
  as a helper.
* water\_bodies: NHDPlus waterbodies loaded directly from the  
  National Seamless using  
  [sf](https://r-spatial.github.io/sf/)  
  read\_sf().
* min\_monthlies: Minimum of the twelve monthly flow estimates for  
  each flowline. Calculated by applying the min function to all the  
  monthly flow collumns from the flowlines attributes.
* remove\_fcodes: NHDPlus feature codes that should be removed.
* crs: A coordinate reference system to perform the analysis in.
* states: A set of US state boundaries loaded from shapefile with  
  sf.
* bbox: A bounding box to limit the analysis to less than the whole  
  country. Created in the sf bbox format then converted in an sfc  
  geometry.

library(sf)

library(dplyr)

# to install nhdplusTools

# devtools::install\_github("dblodgett-usgs/nhdplusTools")

# First we will use a nhdplusTools to load up the national seamless geodatabase.

nhdplusTools::nhdplus\_path("nhdplus\_data/NHDPlusV21\_National\_Seamless.gdb")

staged\_data <- nhdplusTools::stage\_national\_data(include = "flowline",

output\_path = "nhdplus\_data/")

flowlines <- readRDS(staged\_data$flowline)

# Now lets read in the waterbodies directly from the national seamless database.

if("waterbodies" %in% scenario) {

water\_bodies <- read\_sf(nhdplus\_path(), "NHDWaterbody")

}

if("filter\_monthly\_flow" %in% scenario) {

monthlies <- which(grepl("QA\_[0-1][0-9]", names(flowlines)))

min\_monthlies <- apply(st\_set\_geometry(flowlines, NULL)[monthlies], 1, min)

}

if("remove\_intermittent" %in% scenario) {

fcodes <- read\_sf(nhdplus\_path(), "NHDFCode")

remove\_fcodes <- fcodes[which(fcodes$Hydrograph == "Intermittent"), ]

}

# http://spatialreference.org/ref/sr-org/epsg-5070/

crs <- st\_crs(5070)

# See: https://www.arcgis.com/home/item.html?id=b07a9393ecbd430795a6f6218443dccc for this file

states <- read\_sf("states\_21basic/states.shp")

# http://bboxfinder.com/#25.284438,-127.265625,43.707594,-94.438477

bbox <- c(-127.265625,25.284438,-94.438477,43.7075949)

names(bbox) <- c("xmin", "ymin", "xmax", "ymax")

class(bbox) <- "bbox"

bbox <- st\_as\_sfc(bbox)

st\_crs(bbox) <- 4326

**Transform and Filter**

In this next step, I transform all the data into a consistent coordinate  
reference system and get everything filtered. This is all the data I  
want to use for the actual nearest neighbor search. Processing includes:  
– states: Transform to analysis coordinate system and remove small  
islands. – flowlines and water\_bodies: Filter out flowlines with min  
monthly flow of 0 and feature codes indicating intermittent. Transform  
to analysis coordinate system and intersect with analysis bounds and  
simplify geometry to remove un-needed data and precision.

At this point, I save the input flowlines and waterbodies to disk for  
use later because the process is destructive and I need to be careful  
how much system memory I’m using.

bbox <- st\_transform(bbox, crs)

states <- states %>%

st\_transform(crs) %>%

st\_cast("POLYGON") %>%

mutate(AREA = st\_area(.)) %>%

filter(AREA > units::set\_units(2500000000, "m^2")) %>%

select(-AREA)

if("filter\_monthly\_flow" %in% scenario) {

flowlines <- flowlines[which(min\_monthlies !=0 ), ]

}

if("remove\_intermittent" %in% scenario) {

flowlines <- flowlines[which(!flowlines$FCODE %in% remove\_fcodes$FCode), ]

if("waterbodies" %in% scenario) {

water\_bodies <- water\_bodies[which(!water\_bodies$FCODE %in% remove\_fcodes$FCode), ]

}

}

flowlines <- flowlines %>%

st\_transform(crs) %>%

st\_simplify(1000) %>%

st\_intersection(bbox)

if("waterbodies" %in% scenario) {

water\_bodies <-

st\_transform(water\_bodies, crs) %>%

st\_buffer(0) %>%

st\_simplify(500) %>%

st\_intersection(bbox)

}

# Save some intermediate artifacts that we'll read back in later.

saveRDS(flowlines, "temp\_flowlines.rds")

if("waterbodies" %in% scenario) saveRDS(water\_bodies, "temp\_water\_bodies.rds")

**Convert to coordinates**

Now that I have all my data in the right coordinate system and have  
removed all the data that I don’t want, I can set up the data for the  
actual nearest neighbor search. This code converts the data to  
coordinate pairs with a set of identifiers for each feature and feature  
part. Some of this code is not used, but is included to demonstrate how  
identifiers work in what comes out of sf::st\_coordinates. The  
“MULTILINESTRING” coordinates have columns X, Y, L1, and L2 where L1 is  
the part number and L2 is the overall feature. The “MULTIPOLYGON”  
coordinates have columns X, Y, L1, L2, and L3 where L1 is for the main  
ring or holes, L2 for each polygon, and L3 for the overall feature. In  
this case, I just extract the identifier for the overall feature and  
could have used that information to track back to the actual water body  
or flowline in particular that I end up with at the end.

if("waterbodies" %in% scenario) wb\_COMID <- water\_bodies$COMID

fl\_COMID <- flowlines$COMID

# Now turn both flowlines and water\_bodies into coordinate pairs only

flowlines <-

st\_cast(flowlines, "MULTILINESTRING") %>%

st\_coordinates()

if("waterbodies" %in% scenario) {

water\_bodies <-

st\_cast(water\_bodies, "MULTIPOLYGON") %>%

st\_coordinates()

}

# Extract the identifier of features from the coordinates

flowlines <- flowlines[,c(1,2,4)] %>%

data.frame() %>%

rename(ID = L2)

if("waterbodies" %in% scenario) {

water\_bodies <- water\_bodies[,c(1,2,5)] %>%

data.frame() %>%

rename(ID = L3)

}

# Switch the new ID column to the "COMID" values from the source data.

if("waterbodies" %in% scenario) {

water\_bodies[["COMID"]] <- wb\_COMID[water\_bodies[["ID"]]]

}

flowlines[["COMID"]] <- fl\_COMID[flowlines[["ID"]]]

# Bind together into one huge set of coordinates.

if("waterbodies" %in% scenario) {

coords <- rbind(water\_bodies, flowlines) %>%

select(-ID)

rm(flowlines, water\_bodies)

} else {

coords <- flowlines %>%

select(-ID)

rm(flowlines)

}

**Search Points**

In this short step, I create my set of all points using the function  
[expand.grid()](https://www.rdocumentation.org/packages/base/versions/3.5.1/topics/expand.grid?tap_a=5644-dce66f&tap_s=10907-287229)  
and the extents of the bounding box I created above. Using the complete  
grid of locations, I convert it to as sf data.frame and remove all  
the points outside the analysis box and outside the states. I then  
convert the set of points inside the analysis area to a data.frame()  
of coordinates.

# Create a set of search locations

search\_bbox <- st\_bbox(bbox)

x <- seq(search\_bbox$xmin, search\_bbox$xmax, 5000)

y <- seq(search\_bbox$ymin, search\_bbox$ymax, 5000)

search <- expand.grid(x,y)

# Convert to sf and intersect with the state boundaries.

search <- st\_as\_sf(data.frame(search), coords = c("Var1", "Var2"), crs = crs) %>%

st\_intersection(states) %>%

st\_intersection(bbox) %>%

st\_coordinates() %>%

data.frame()

**Plot function**

In order to create the animated gif, I needed to be able to create a  
similar plot over and over. So I created a function to do the job. This  
function creates an output file name, sets up a png, gets the data ready  
to plot, then plots a few layers using base R graphics and  
[plot.sf](https://r-spatial.github.io/sf/reference/plot.html).

# Function to plot results in a .png

plot\_result <- function(search, radius, crs, bbox, states) {

fname <- stringr::str\_pad(paste0(as.character(radius), ".png"), 10, "left", "0")

png(fname,

width = 1000, height = 800)

result <- st\_as\_sf(search, coords = c("X", "Y"), crs = crs)

plot(bbox, main = paste("Distance to nearest water:", radius, "meters"))

plot(states$geometry, add = TRUE)

plot(result$geometry, pch = 19, add = TRUE)

dev.off()

}

**Run the nearest neighbor search**

Finally, we are ready to run the analysis. To me, while loops are the  
forbidden fruit of software development, but sometimes you just don’t  
know how many times you need to run a loop! In this case, we run the  
[RANN::nn2()](https://www.rdocumentation.org/packages/RANN/versions/2.6/topics/nn2?tap_a=5644-dce66f&tap_s=10907-287229)  
function which returns the index of the matching nearest neighbor  
(nn.idx), if one is found. If no nearest neighbor is found in the  
search radius, it returns nn.idx of 0. So in every loop, I only keep  
matches where nn.idx == 0 returns true. I plot the filtered set to a  
png and increment the radius up for the next loop. The if/else block at  
the bottom of this step just slows down the rate of increase of the  
radius as we approach only one location left to avoid overshooting and  
missing the last one!

Once the while loop finishes, I pass the list of pngs created to  
[gifski](https://www.rdocumentation.org/packages/gifski/versions/0.8.3/topics/gifski?tap_a=5644-dce66f&tap_s=10907-287229)  
a great little package for creating animated gifs. Finally I printed out  
the lat/lon of the location found so I could go look at the location in  
google.

radius <- 0

num\_left <- nrow(search)

while(num\_left > 1) {

num\_left <- nrow(search)

# This is where the magic happens.

matched <- RANN::nn2(coords[,1:2],

search,

k = 1,

searchtype = "radius",

radius = radius) %>%

data.frame()

# This while loop is destructive. Only keep unmatched.

search <- filter(search, matched$nn.idx == 0)

plot\_result(search, radius, crs, bbox, states)

if(num\_left > 50) {

radius <- radius + 1000

} else if (num\_left > 3) {

radius <- radius + 500

} else {

radius <- radius + 100

}

}

rm(coords)

gifski::gifski(list.files(pattern = "\*0.png"))

# Where in the world is the result?

result <- st\_as\_sf(search, coords = c("X", "Y"), crs = crs)

print(st\_transform(result$geometry, 4326))

**Plot the result location**

Finally, I plot up the results at a local scale. Steps for this include:  
– I want the original data I started with, so I just load it back from  
disk. – I set up an area for the plot by buffering around the result  
location by 150km and converting the result to a simple box. –  
Increasingly, I like using the web mercator projection for visualization  
because it’s familiar and compatible with web-map tiles. – I project all  
the data to my plotting projection and subset it and simplify the  
geometry to speed things up and not overwhelm plot() with data for the  
whole country. – I grab a background map from google with ggmap::getmap  
to be passed to the bgMap input of plot.sf(). – I plot each layer I  
want into a png. – Finally I delete the png files and temporary geometry  
I wrote to disk.

# Load geospatial data again.

flowlines <- readRDS("temp\_flowlines.rds")

if("waterbodies" %in% scenario) water\_bodies <- readRDS("temp\_water\_bodies.rds")

# Set up a plot area around our result.

plot\_area <- st\_buffer(result$geometry, 150000) %>%

st\_bbox() %>%

st\_as\_sfc()

st\_crs(plot\_area) <- crs

# Use 3857 (web mercator) for visualization.

plot\_proj <- 3857

plot\_area <- st\_transform(plot\_area, plot\_proj)

# Subset data to an area that satisfies plot\_area.

data\_area <- st\_bbox(plot\_area) %>%

st\_as\_sfc()

st\_crs(data\_area) <- st\_crs(data\_area)

data\_area <- st\_transform(data\_area, crs)

if("waterbodies" %in% scenario) {

local\_water <- st\_intersection(water\_bodies, data\_area) %>%

st\_simplify(1000) %>%

st\_transform(plot\_proj)

}

local\_flowlines <- st\_intersection(flowlines, data\_area) %>%

st\_simplify(5000) %>%

st\_transform(plot\_proj)

bgmap <- plot\_area %>%

st\_transform(4326) %>%

st\_bbox() %>%

setNames(c("left", "bottom", "right", "top")) %>%

ggmap::get\_map(zoom = 7)

# write out a png of the local area.

png("furthest\_water.png", width = 1024, height = 1024)

par(omi = c(0,0,0,0), mai = c(0,0,0,0))

plot(plot\_area, col = NA, border = NA, xaxs = 'i', yaxs = 'i', bgMap = bgmap)

plot(st\_transform(states$geometry, plot\_proj), add = TRUE)

plot(st\_transform(result$geometry, plot\_proj), col = "red", cex = 3, lwd = 4, add = TRUE)

plot(local\_flowlines$Shape, add = TRUE, col = "blue")

if("waterbodies" %in% scenario) plot(local\_water$Shape, add = TRUE, col = "azure2")

dev.off()

unlink("temp\_water\_bodies.rds", force = TRUE)

unlink("temp\_flowlines.rds", force = TRUE)

unlink(list.files(pattern = "\*0.png"), force = TRUE)

That’s it! If you made it this far, thanks for taking the time! I hope  
this was helpful one way or another.